VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Monitorovací systém pro dobíjecí stojany elektromobilů

Monitoring system for recharging electric vehicles stands

2015

Kateřina Káňová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Kateřina Káňová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Monitorovací systém pro dobíjecí stojany elektromobilů
Monitoring system for recharging electric vehicles stands

Zásady pro vypracování:

V současné době se na fakultě vyvíjejí dobíjecí stojany pro elektromobily. Tyto stojany umožňují dálkovou správu. Cílem práce bude vytvořit webovou aplikaci, který bude spravovat dobíjecí stojany pro elektromobily. Během řešení práce postupujte podle těchto bodů.

1. Seznamte se s aktuálním systémem a s požadavky kladenými na nový systém.
2. Vytvořte analýzu a návrh implementace nového systému.
3. Naimplementujte aplikaci s ohledem na moderní webové technologie.
4. Vytvořte dokumentaci k vytvořenému systému.

Seznam doporučené odborné literatury:

[1] Robert C. Martin, Čistý kód, Computer Press 2009
[2] Patterns of Enterprise Application Architecture, Fowler Martin, 2002
[3] Real World Java EE Patterns Rethinking Best Practices, Bien Adam, Press Adam Bien 2012
[4] Real World Java EE Night Hacks Dissecting the Business Tier, Press Adam Bien, 2011
[5] The Java EE 7 Tutorial, Jendrock Eric, http://docs.oracle.com/javaee/7/tutorial/doc/home.htm

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lumír Návrat**

Konzultant bakalářské práce: Ing. Zdeněk Slanina, Ph.D.

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 5. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 5. května 2015 .....

Ráda bych poděkovala především vedoucímu této práce, panu Ing. Lumíru Návratovi, za vstřícný přístup a čas, který mé práci věnoval. Dále bych chtěla poděkovat panu Ph. D. Slaninovi za konzultace k systému. Nemenší díky patří i mým rodičům, kteří mě ve studiu po celou dobu plně podporovali.

Abstrakt

Vzhledem k rozvoji automobilových technologií se na našich cestách začínají objevovat elektromobily. Z toho důvodu byla vybudována síť dobíjecích stanic, která se dále rozšiřuje.

Aby bylo možné data z těchto stanic přehledně zpracovávat, vznikla webová aplikace, která umožňuje jak správu uživatelů těchto stanic, tak dat, týkajících se samotného dobíjení.

Obsahem této práce je zdokonalení systému, který pro tyto účely již sloužil, a využití nových technologií, které tento systém podpoří. Práce popisuje vývoj tohoto systému od samotných požadavků až po implementaci.

Klíčová slova: bakalářská práce, Java, JSF, EJB, JPA, elektromobily, dobíjecí stojany

Abstract

According to the progress of car technology we can see electric cars on our roads. This is a reason why a new net of recharging stations was bilt there and why it is more widened. For compass of data processing at this stations a new web application, witch provide both an administration of users of these stations and data related to recharging itself, was lanched .

The contents of my work are improvements of the system used for these aims and the use of new technogies for supporting of this system. My work describes this system from requirements themselves to an implementation.

Keywords: bachelor thesis, Java, JSF, EJB, JPA, electric cars, charging stand

Seznam použitých zkratek a symbolů

EJB	–	Enterprise JavaBeans
JPA	–	Java Persistence API
JSF	–	Java Server Faces
JSP	–	Java Server Pages
CDI	–	Contexts and Dependecy Injection
REST	–	Representational State Transfer
DAO	–	Data Access Object

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Vzhledem k rozvoji automobilových technologií se na našich cestách začínají objevovat elektromobily. Z toho důvodu byla vybudována síť dobíjecích stanic, která se dále rozšiřuje.

Aby bylo možné data z těchto stanic přehledně zpracovávat, vznikla webová aplikace, která umožňuje jak správu uživatelů těchto stanic, tak data týkající se samotného dobíjení.

Obsahem této práce je zdokonalení systému, který pro tyto účely již sloužil, a využití nových technologií, které tento systém podpoří. Práce popisuje vývoj tohoto systému od samotných požadavků až po implementaci.

V rámci ČR existuje již několik stanic, sloužících k dobíjení elektromobilů. Vzniklý informační systém umožňuje přehlednou správu dat z těchto stanic pomocí webového rozhraní. Data z dobíjecích stojanů jsou odesílána do aplikace, která je zpracovává a ukládá do databáze. Nad tou se nachází webová aplikace umožňující správu uživatelů, zobrazování stavu stojanů, přidávání či odebírání stanic a další.

Cílem této práce bylo použít novější technologie a vylepšit architekturu systému tak, aby byla příjemnější pro uživatele, a zároveň byla flexibilnější s ohledem na pružnost kódu a možnost přidávání nových funkcí či jiné změny v systému v budoucnosti.

Kapitola 2 popisuje požadavky, které jsou na tento systém kladeny. Popisuje jak požadovanou funkčnost systému, tak další požadavky, týkající se například vzhledu aplikace.

Kapitola 3 se věnuje analýze těchto požadavků. Je rozdělena na analýzu aplikace, která se zabývá převážně funkčností, dále je zde uvedena analýza dat, které budou v systému dominovat. Stručně je zde popsána i analýza změn webového rozhraní.

Kapitola 4 už uvádí konkrétní návrhy a implementace, které byly v aplikaci použity, a neméně popisuje také využité technologie.

2 Specifikace požadavků

V této kapitole se pokusím specifikovat požadavky kladené na systém dobíjecích stanic a stanovit funkcionalitu, kterou by měl tento systém umožňovat. Na každý systém jsou kladeny požadavky, které se v průběhu vývoje systému dále rozvíjejí a přibývají k nim nové. V této kapitole se pokusím shrnout všechny požadavky, kterými se vývoj systému řídil.

2.1 Funkční požadavky

Vzhledem k tomu, že dosavadní systém již nebyl vyhovující, je potřeba vytvořit nový, který bude splňovat funkčnost, a jehož implementace bude efektivnější. Tento systém má obstarávat přehled o stojanech a stanicích. Každý, kdo se do systému přihlásí, uvidí, zda je stojan v provozu nebo zda se z něj dobíjí. Mimo to bude tento informační systém umožňovat i zobrazení seznamu uživatelů či zákazníků a jejich aktivitu. Systém bude taky umožňovat tištění dokladů pro jednotlivá dobíjení.

Celý proces dobíjení by měl z pohledu zákazníka vypadat tak, že zákazník přijede s elektromobilem k dobíjecímu stojanu a pomocí karty se přihlásí. Stojan mu umožní dobít si automobil. Po dokončení dobíjení se zákazník odhlásí.

Z hlediska správce stanice by systém měl vypadat tak, že umožní správu stojanů dané stanice, zobrazení historie stojanů a případný tisk dokladů k jednotlivým dobíjením prováděných u těchto stojanů. Správce stanice by neměl mít přístup k jiné stanici, než je mu přidělena. Data by ale měla být přístupná i majiteli, který má přehled nad více stanicemi a může si zobrazovat seznamy zákazníků a informace o nich. Největší práva by měl mít administrátor, který bude moci navíc přidávat a odebírat uživatele této aplikace.

Rozhraní pro komunikaci mezi uživateli a systémem budou webové stránky. Tento požadavek vyplývá z toho, že bude nutné do systému přistupovat z různých míst, a to nejen ze samotných stanic. Aplikace bude se stojany komunikovat a změny bude zobrazovat v reálném čase.

Vstupy budou pocházet ze dvou zdrojů. Ze stojanu se budou ukládat data do databáze, která budou obsahovat informace o konkrétních stanicích a stojanech. Také se budou ze strany stojanu získávat data o dobíjení, která budou obsahovat konkrétního zákazníka a informace o dobíjení (například čas, cenu). Dalšími vstupy budou data přímo z aplikace, skrze kterou bude možné editovat správu nad dalšími stanicemi a stojany, uživateli a zákazníky. Výstupy budou prezentovány převážně aplikační části, která bude sloužit k zobrazování dat a jejich správě. Mimo to bude umožňovat tisk dokladu o dobíjení.

Vzhledem k tomu, že cílem tohoto systému je mimo jiné i rozšíření do dalších zemí, je kladen důraz i na možnost přepínání jazyků, které budou používány ve webovém rozhraní.

2.2 Technická specifikace

Webové rozhraní bude uživatelsky přívětivé s líbivým moderním designem. Bude navrhnuto tak, aby ovládání bylo intuitivní a přehledné.

Komunikace mezi stojanem a aplikací bude bezestavová.

Aplikace by měla zobrazovat aktuální data. V případě dobíjení budou data na webových stránkách aktualizovaná okamžitě - tzn. nebudou čekat až na reload stránky ze strany uživatele.

Zatím se počítá s desítkami uživatelů a stovkami zákazníků. Kapacita databáze bude pro začátek stačit menší, protože v databázi budou uložena pouze textová data.

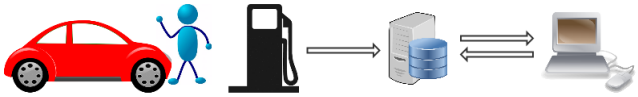
Po aplikaci se požaduje, aby byla přístupná nepřetržitě. Největší nápor se předpokládá přes den, kdy budou zákazníci dojíždět a obsluha bude tisknout doklady přístupné z aplikace.

3 Analýza

Tato část se bude zabývat analýzou vycházející s daných požadavků specifikovaných v předchozí kapitole. V první části bude popsáno chování aplikace, v další části pak bude uvedena analýza datové části určené pro ukládání dat.

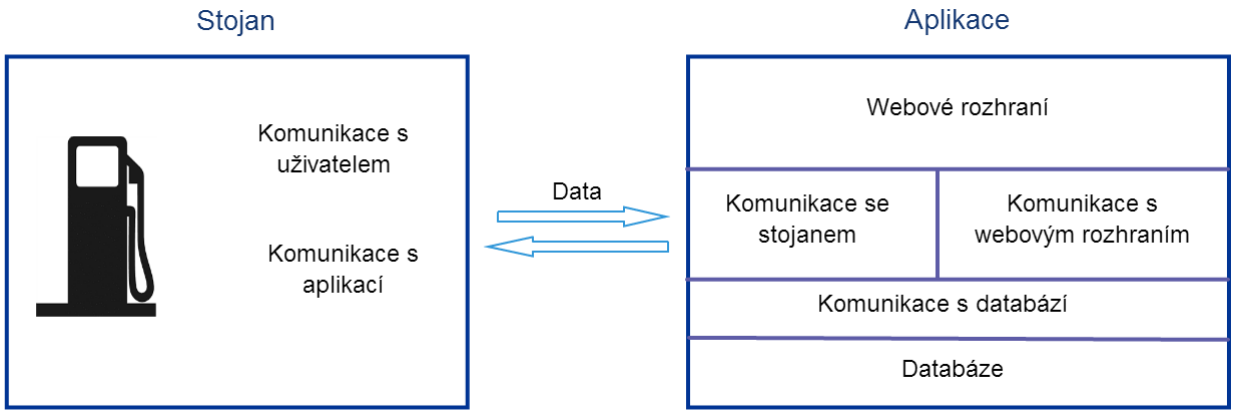
3.1 Analýza aplikace

Dle požadavků lze celý systém rozdělit na část tvořenou stojanem, kam zákazník přijíždí dobíjet svůj elektromobil, a na aplikaci, která umožňuje správu dat.



Obrázek 1: Schéma fungování systému pro dobíjení elektromobilů

Z hlediska funkcí jednotlivých částí systému bude každá část obsahovat funkční prvky uvedené na obrázku 2.

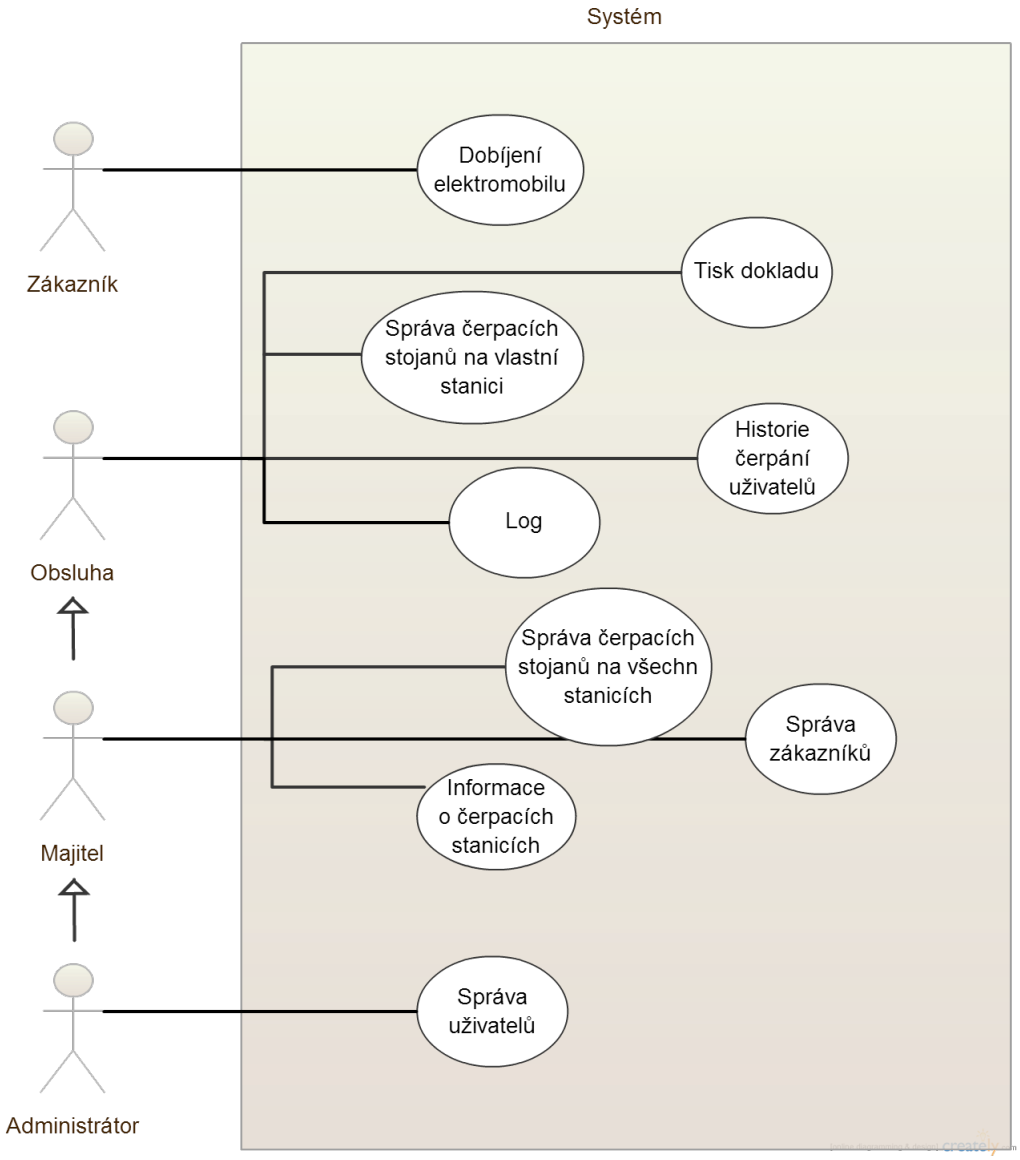


Obrázek 2: Logické schéma rozložení základní funkčnosti systému

3.1.1 Uživatelé

Z hlediska správy dat bude do aplikace možno vstupovat v několika rolích, které budou určovat, jaká funkcionalita bude přihlášenému uživateli povolena. UseCase diagram 3 zachycuje rozvrstvení uživatelů a jejich práv v aplikaci.

Stanice, uživatelé i zákazníci mohou být přidáváni, mazáni i upravováni. Stojany mohou být také přidávány, mazány či upravovány, navíc mohou být přiřazovány k jednotlivým stanicím.



Obrázek 3: UseCase diagram pro uživatele systému

Na základně tohoto diagramu můžeme rozepsat jednotlivé role:

- **Administrátor**
  - zobrazení čerpacích stojanů na jednotlivých stanicích
  - historie čerpání uživatelů
  - seznamy zákazníků a uživatelů
  - informace o čerpacích stanicích
  - log
- **Majitel**
  - zobrazení čerpacích stojanů na jednotlivých stanicích
  - historie čerpání uživatelů
  - seznamy uživatelů
  - informace o čerpacích stanicích
  - log
- **Obsluha**
  - zobrazení čerpacích stojanů pouze na vlastní stanici
  - historie čerpání uživatelů
  - log
- **Zákazník**
  - dobíjení elektromobilů po přihlášení ke stojanu

3.1.2 Analýza komunikace

Tento systém je tvořen několika částmi, které mezi sebou komunikují.

Jednou z částí, která bude komunikovat se serverem, je stojan. Pokud se uživatel připojí, stojan musí vysílat data na server. Přenášená data budou obsahovat identifikaci uživatele, informace o době nabíjení a odebraných kilowatech.

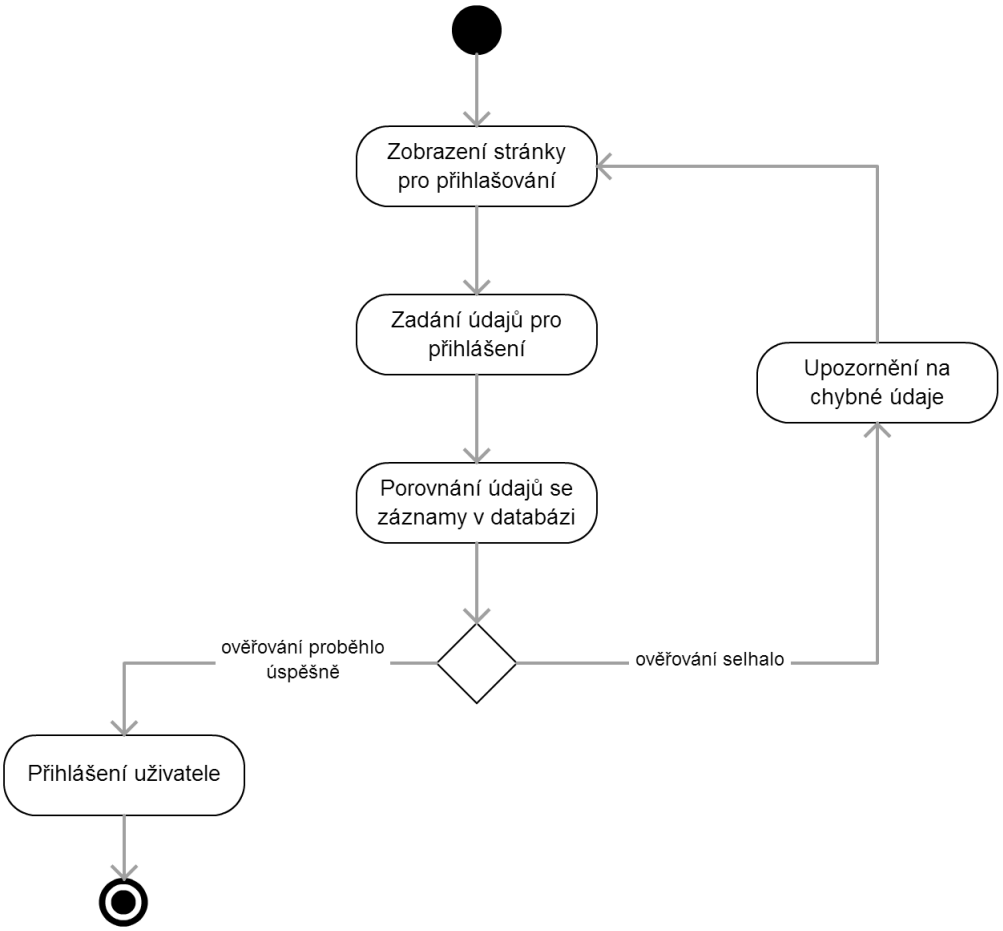
Další částí, která bude zprostředkovávat komunikaci, bude webové rozhraní, které bude sloužit pro komunikaci s uživatelem. Toto rozhraní bude reagovat jednak na podněty ze strany klienta, ale také musí být schopno reagovat na informace ze strany stojanu a aktualizovat data ve webovém rozhraní v reálném čase (nečekat na to, až uživatel stránku obnoví).

3.1.3 Aktivitní diagramy

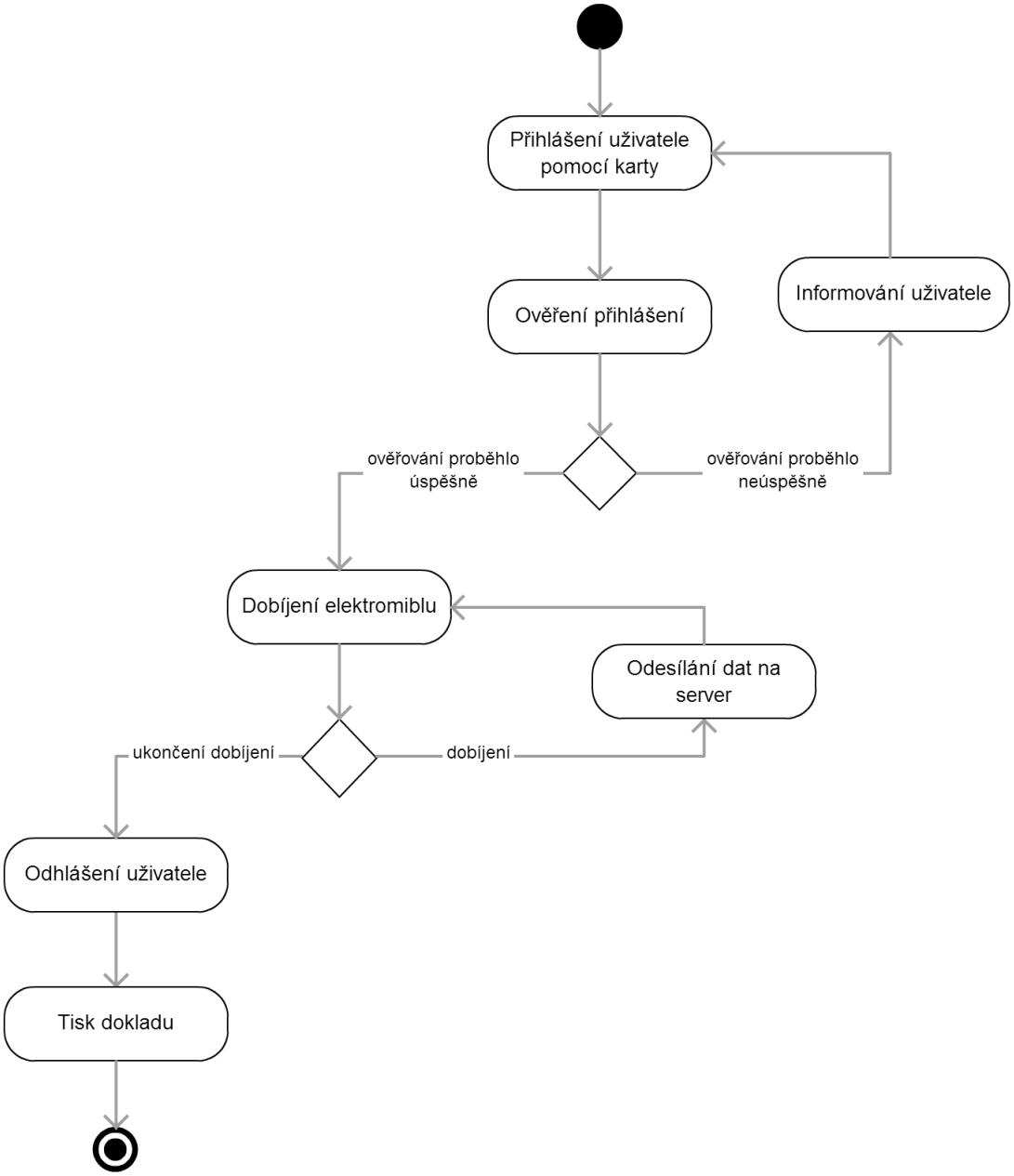
V této kapitole se nacházejí aktivitní diagramy znázorňující chování aplikace.

V diagramu 4 je znázorněn průběh přihlašování uživatele. Vstupní údaje se porovnávají s údaji o uživatelích, které jsou uloženy v databázi.

V diagramu 5 je zachycen průběh dobíjení elektromobilu ze stojanu. Uživatel se přihlašuje pomocí karty. Přihlašovací údaje jsou porovnány s údaji v databázi. Po úspěšném přihlášení může uživatel dobíjet elektromobil. Data o dobíjení jsou zasílána na server. V případě, že nějaký uživatel má zobrazenou stránku s tímto stojanem, v reálném čase vidí, že z tohoto stojanu probíhá dobíjení. Po ukončení dobíjení je uživatel odhlášen.



Obrázek 4: Aktivitní diagram pro přihlašování uživatelů



Obrázek 5: Aktivitní diagram pro dobíjení elektromobilu

3.1.4 UseCase Scénáře

V této kapitole uvidíme některé UseCase scénáře, které blíže popíší chod systému. Další UseCase scénáře jsou uvedeny v přílohách.

Priorita znamená, jak důležité je, aby tento UseCase fungoval. Hodnota Priority nabývá od 1 do 10, kdy 1 je nejmenší priorita a 10 nejvyšší.

První uvedený UseCase Scénář se zabývá Dobíjením elektromobilu. Popisuje proces od přihlášení uživatele až po jeho odhlášení ze systému.

ID	UC1
Název	Dobíjení elektromobilu
Popis	Zákazník přijede s elektromobilem ke stojanu, přihlásí se, připojí elektromobil ke stojanu a probíhá dobíjení. Poté se uživatel odhlásí ze systému.
Primární aktor	Zákazník
Prekondice	Zákazník musí mít přístupová práva ke stojanům.
Postkondice	Zákazník dobije elektromobil a data se uloží do databáze.
Hlavní scénář	1. Zákazník se přihlásí do systému. 2. Aplikace uloží informace o přihlášení. 3. Stojan začne odesílat data o nabíjení. 4. Aplikace ukládá data. 5. Zákazník ukončí dobíjení. 6. Stojan odešle data a odhlásí uživatele. 7. Aplikace uloží informace o uživateli (id), stojanu (id stanice, stojanu, výstup), délku a cenu dobíjení.
Rozšíření	4a Uživatel má zobrazenou webovou stránku s daným stojanem. 4a1 Aplikace aktualizuje data na stránce.
Četnost užití	Denně
Priorita	10

Tabulka 1: UseCase Scénář - Dobíjení elektromobilu

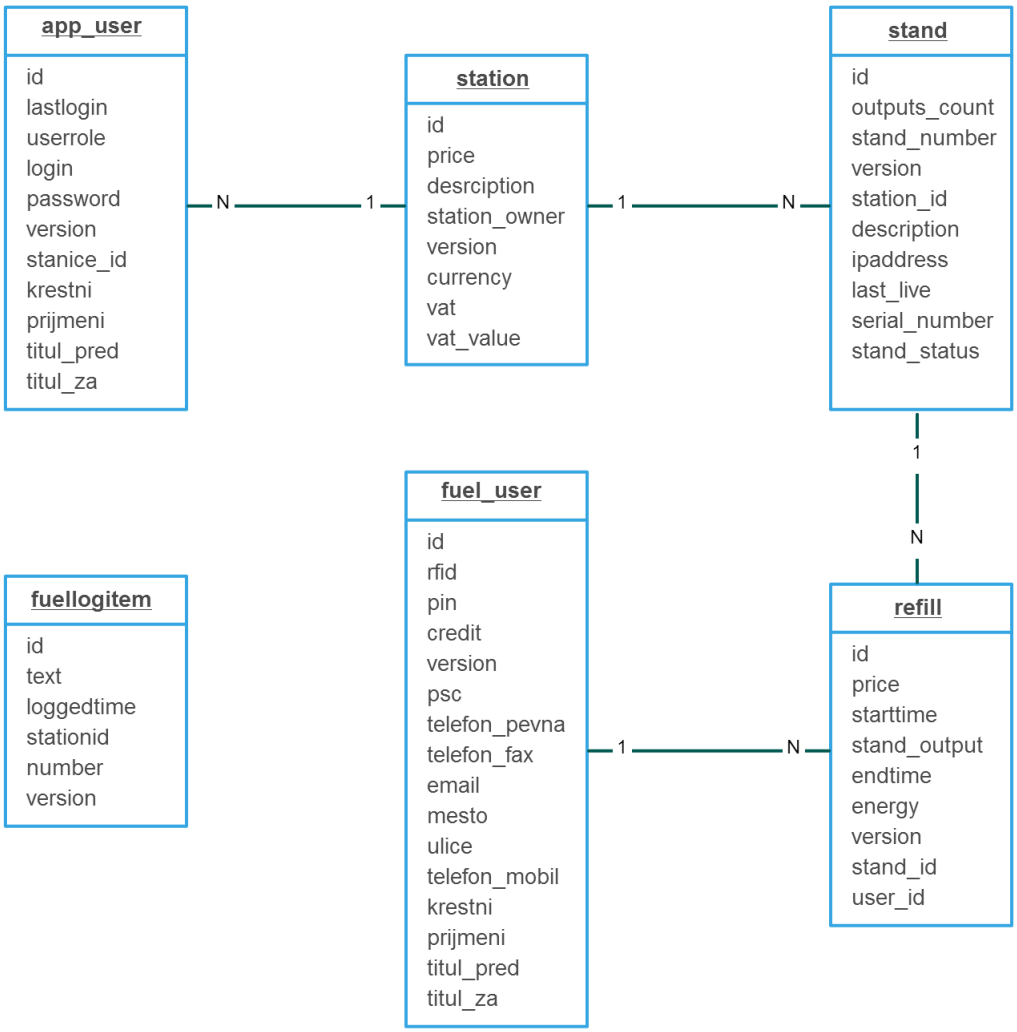
Druhý UseCase Scénář popisuje správu stojanů na stanicích.

ID	UC2
Název	Správa čerpacích stojanů na všech stanicích
Popis	Uživatel zvolí stanici a stojan, provede úpravy a ty se uloží do databáze.
Primární aktér	Majitel
Prekondice	Uživatel musí mít dostatečná práva a přístupy.
Postkondice	Databáze je aktualizovaná a obsahuje upravená data.
Hlavní scénář	1. Uživatel se přihlásí do systému. 2. Aplikace zobrazí seznam stanic. 3. Uživatel zvolí stanici. 4. Aplikace zobrazí seznam stojanů na dané stanici. 5. Uživatel zvolí stojan. 6. Aplikace zobrazí informace o zvoleném stojanu. 7. Uživatel upraví a uloží informace: vlastník, popis, cena, měna, počet stojanů, počet výstupů, ip adresa, sériové číslo, komentář. 8. Aplikace uloží data do databáze.
Alternativní scénář	7a Uživatel neupraví nebo stornuje úpravy informací. 7a1 Aplikace v databázi zanechá původní data
Četnost užití	Měsíčně
Priorita	10

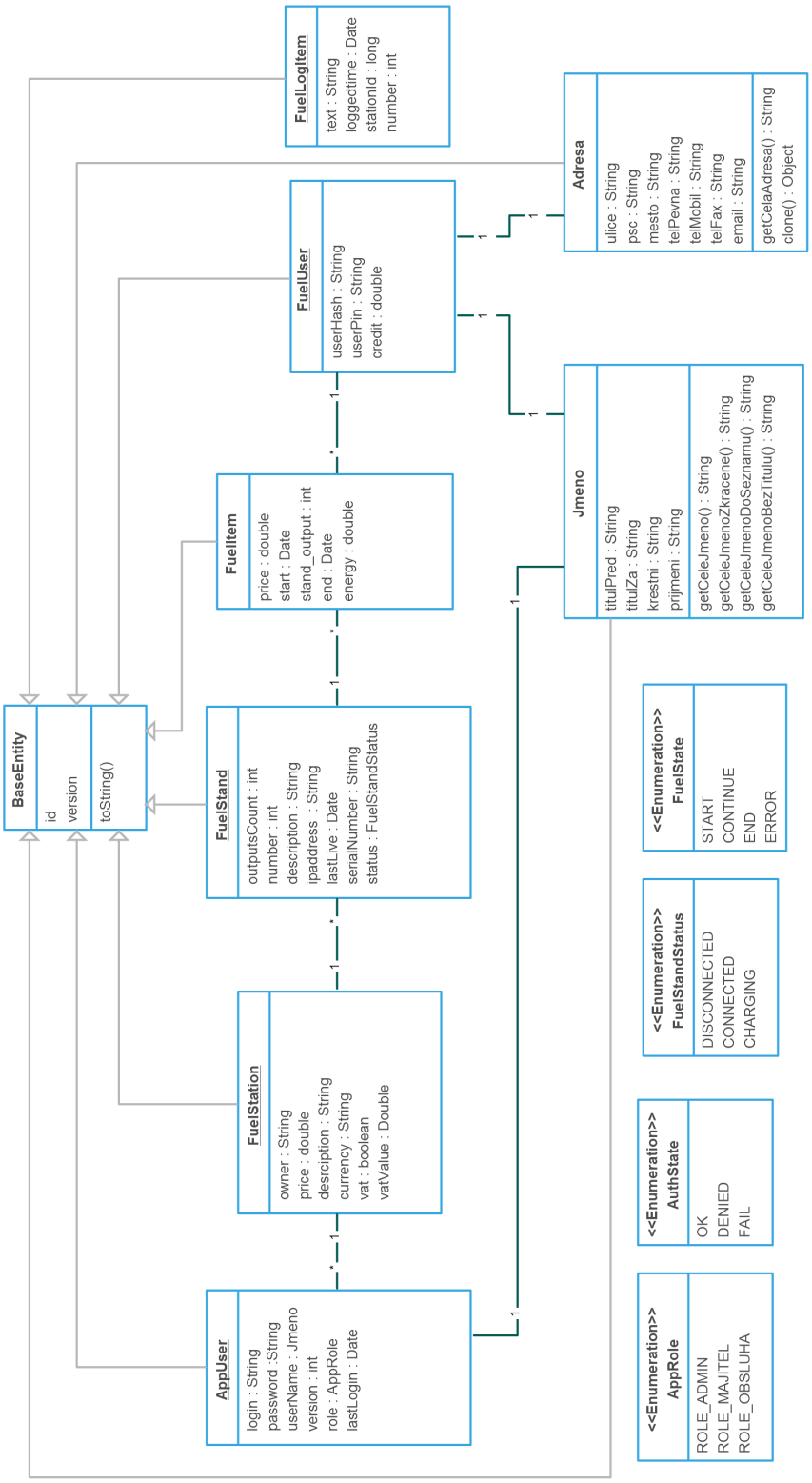
Tabulka 2: UseCase Scénář - Dobíjení elektromobilu

3.2 Analýza dat

Datová analýza popisuje strukturu databáze, jednotlivé tabulky s jejich atributy a vztah mezi tabulkami. Grafické znázornění použité databáze můžeme vidět na obrázku 6. Některé tabulky obsahují více logických celků najednou, takže v programu je vhodné je rozdělit. Logické rozdělení ukazuje třídní diagram 7. Jak můžeme vidět, třídy Jmeno a Adresa jsou v databázi součásti tabulky app_user a fuel_user.



Obrázek 6: Schéma databáze



Obrázek 7: Třídní diagram

3.2.1 Lineární zápis

Lineární zápis: **Název_tabulky**(***primární klíč***, *cizí klíč*, atributy).

app_user(***id***, *stanice_id*, , lastlogin, userrole, login, password, version, krestni, prijmeni, titul_pred, titul_za)

fuel_user(***id***, rfid, pin, credit, version, psc, telefon_pevna, telefon_fax, email, mesto, ulice, telefon_mobil, krestni, prijmeni, titul_pred, titul_za)

fuellogitem(***id***, text, loggedtime, stationid, number, version)

refill(***id***, *stand_id*, *user_id*, price, startime, stand_output, endtime, energy, version)

stand(***id***, *station_id*, outputs_count, stand_number, version, description, ipaddress, last_live, serial_number, stand_status)

station(***id***, price, desrciption, station_owner, version, currency, vat, vat_value)

USER_STATION(app_user, station) N:1
STANDS(station, stand) 1:N
FUELING(stand, refill) 1:N
USER_FUELING(fuel_user, refill) 1:N

3.2.2 Popis tabulek

V databázi se nachází dvě tabulky reprezentující uživatele. Jendou je app_user a druhou je fuel_user.

app_user
: Tato tabulka reprezentuje uživatele aplikace. Má nastavenou vlastní roli, přihlašovací údaje a jméno. Má také svoji stanici, která je zvolena jako výchozí při přihlášení do systému. Pokud jde o roli uživatele Obsluha, pak je to také jediná stanice, ke které má uživatel přístupová práva. Do této tabulky se eviduje také poslední login uživatele do aplikace.

fuel_user
: Oproti předchozímu typu uživatele, tento zastupuje zákazníky. Jedná se tedy o ty, kteří si přijedou ke stojanu dobít svůj elektromobil. Tato tabulka obsahuje adresu a kontaktní údaje na tohoto uživatele. Mimo to v ní najdeme také informace o kreditu a uživatelský hash.

stand
: Jak již jméno napovídá, v této tabulce jsou uloženy stojany. Každý stojan má pro lepší identifikaci vlastní číslo. Tabulka stojanu obsahuje také počet výstupů k dobíjení. Stojan má v databázi i svůj status, který označuje, zda je připojen a komunikuje s aplikací. Každý stojan také obsahuje vazbu na stanici, ke které patří. Ta je reprezentována jako id stanice.

station
: Stanice obsahuje krátký popis a vlastníka, ke kterému patří. Mimo to uchovává i informace o měně, ve které se na této stanici platí, a cenu za jeden kilowat.

refill
: Tato tabulka uchovává informace o jednotlivých dobíjeních. Obsahuje odkaz na uživatele, který je ke stojanu přihlášen, a identifikaci stojanu a výstupu, ze kterého dobíjení probíhá. Také v této tabulce nalezneme čas začátku a konce nabíjení včetně ceny, kterou zákazník zaplatí a množství energie, kterou odebral.

fuellogitem
: Tato tabulka obsahuje informace o aktivitách na stanicích, jako je spuštění stanice nebo přihlášení uživatele.

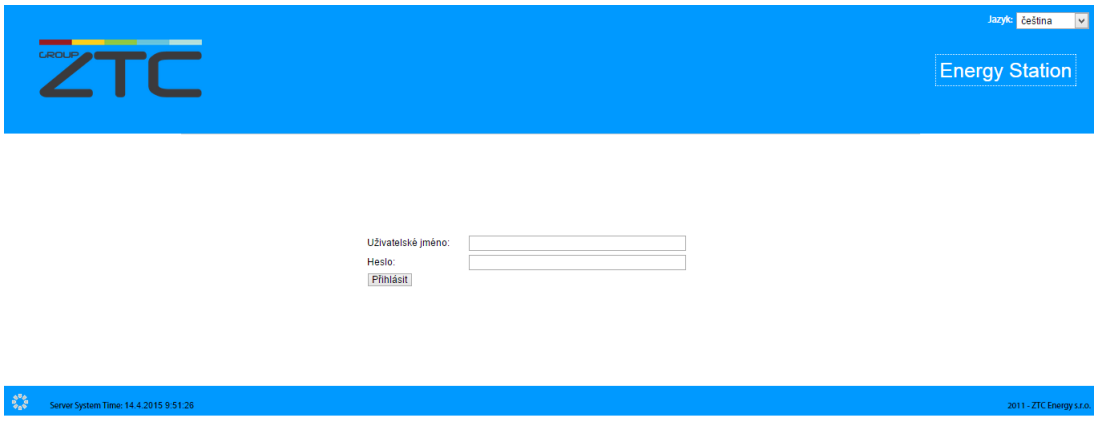
3.3 Analýza webového rozhraní

Jedním z požadavků na webové rozhraní bylo změnit a modernizovat design, aby se staly pro uživatele přívětivější. Také je cílem zjednodušit orientaci na těchto stránkách, což bylo jedním z hlavních důvodů nového designu.

Původní grafickou podobu můžeme vidět na obrázku 8. Ta je nevyhovující z několika důvodů.

Hlavní logo stránek již nebude ZTC, ale nahradí jej samostatné logo systému, případně logo VŠB. Tento vzhled by také mohl na uživatele působit stroze a tak by je mohl demotivovat k další práci s tímto systémem.

Návrh nového designu dodá osoba pověřená konzultantem práce.



Obrázek 8: Starý design

4 Návrh a implementace systému

V této části textu bych se ráda věnovala konkrétnímu návrhu a implementaci systému. V této kapitole se také objeví technologie, které byly pro implementaci použity.

Hlavním jazykem této aplikace je Java, a to převážně z důvodu přenositelnosti na různé používané platformy.

Technologie dominující v tomto systému jsou technologiemi standaru Java EE 7, a to JPA, EJB a JSF. Pro konkrétní webové stránky jsou využity PrimeFaces, které nahrazují původně používané IceFaces.

Aplikačním serverem tohoto systému byl původně GlassFish a jeho použití zůstalo zachováno. Je jedním ze serverů, které podporují Javovskou technologii EJB, a také je pro projekt důležité, že je volně dostupný. Z těchto důvodů byl GlassFish pro tuto aplikaci vhodným kandidátem.

Podkapitoly této části jsou věnovány struktuře aplikace a konkrétním technologiím, ale naleznete zde i ukázky kódů a příklady využití zmíněných technologií v programové části.

4.1 Použité technologie

Nejdříve bych ráda popsala technologie, které byly použity při zpracování aplikace.

4.1.1 Java Persistence API

Java Persistence API (dále JPA) poskytuje POJO model [10] pro objektově-relační mapování. Její použití se ale neomezuje pouze na EJB komponenty.

K využití JPA se používají Entity. Jde o třídy, které reprezentují objekty reálného světa a jsou mapovány na tabulky nebo jejich části z databáze. Aby se třída mohla stát entitou a JPA s ní správě pracovalo, musí splňovat určité podmínky a obsahovat správné anotace [8]. Jednou z možností je používání anotace @Entity, která byla použita i v tomto projektu. Příklad entity vidíme v ukázce kódu 1

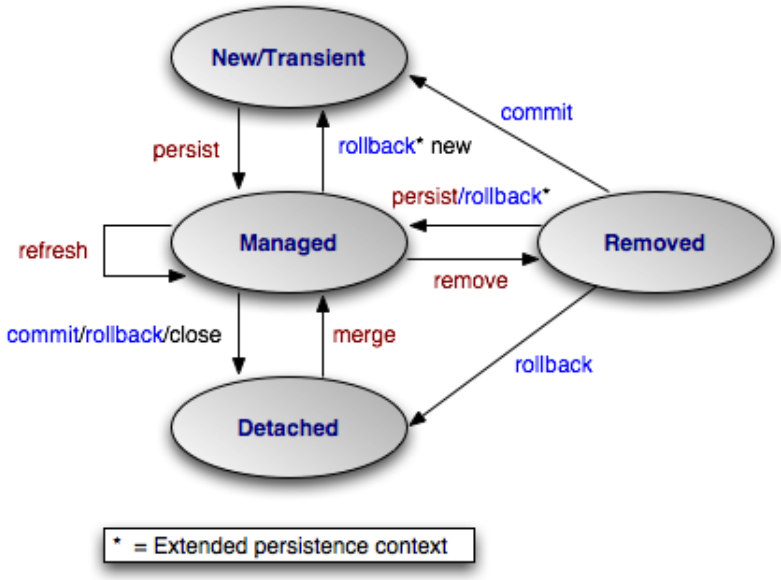
Životní cyklus entity je znázorněn na obrázku 9. Všechny entity jsou spravovány EntityManagerem, díky kterému se nacházejí v určitém stavu. Výchozí stav entity je New/Transient.

Mezi jednotlivými entitami mohou existovat vazby 1:1 1:N a N:M.

Kromě využívání CRUD operací, které umožňuje EntityManager, je ale možné využívat i vlastních funkcí psaných v jazyce Java Presistence Query Language JPQL. JPQL je jazyk podobný SQL, který má ale pár výhod navíc - jednak nezáleží na tom, nad jakou databází funguje, ale také umožňuje objektový přístup. Jednotlivé parametry v dotazovacím řetězci jdou nastavit z proměnných, čímž se dá předejít SQLInjection. Jako ukázku jsem si dovolila uvést jednoduchý select s pomocí JPQL, ve kterém navíc využívám generické typy. V příkladu kódu 2 vidíme využití externího parametru id, který se nastavuje pomocí metody .setParameter. Zde jsem použila .getSingleResult, protože očekávám, že mi dotaz vrátí pouze jeden záznam. V případě, že metoda vrátí kolekci, se používá .getResultList().

```
@Entity
@Table(name = "app_user")
public class AppUser extends BaseEntity implements Serializable {
    @Embedded
    private Jmeno userName;

    @Version
    private int version;

    @Enumerated(EnumType.STRING)
    @Column(name = "userrole", nullable = false)
    AppRole role;

    @ManyToOne
    FuelStation stanice;

    @Temporal(javax.persistence.TemporalType.TIMESTAMP)
    private Date lastLogin;

     ...
}
```

Výpis 1: Ukázka entitní třídy



Obrázek 9: Životní cyklus entity ze zdroje [5]

```
@Override
    public <T extends BaseEntity> T selectOne(Class<T> myClass, Long myId) {
        String selectString = "select class from " + myClass.getName() + " class where class.
            id = :id";
        T result = (T) entityManager.createQuery(selectString, sth)
                .setParameter("id", myId)
                .getSingleResult();
        return result;
    }
```

Výpis 2: Ukázka metody select pomocí JPQL

4.1.2 JavaServer Faces

JavaServer Faces (dále JSF) je technologie, která umožňuje tvoření server-side uživatelských rozhraní. Tato technologie obsahuje UI komponenty, které tvoří dynamickou část stránky. Tyto komponenty mohou měnit svůj stav, obstarávat události a vstupy, definovat navigaci stránek a zlepšovat přístup.

JSF jsou navrženy tak, že používají standardní existující UI a webové koncepty, čímž nenutí developery k používání jiného značkovacího jazyku. UI komponenty JSF obsahují funkcionalitu, neurčují ale vzhled, takže jsou použitelné na různých klientských zařízeních.

Cílem JSF bylo jednoduše oddělit aplikační logiku a presentační vrstvu, ale zároveň zjednodušit jejich propojení.

Samotné JSF značky jsou definovány v knihovnách. JSF jsou rozděleny do jmenných schémat, které musíme definovat v souboru pomocí xmlns, jak lze vidět v ukázce 4.1.2.

```
xmlns:h="http://java.sun.com/jsf/html"
xmlns:p="http://primefaces.org/ui"
```

Jako ukázku nějakého JSF kódu jsem zvolila obyčejné tlačítko, viz ukázka 3. Můžeme si všimnou jednoduchého použití komponenty custBean, která se používá pomocí tzv. binding. Pak se lze objektovým přístupem dostat k jednotlivým metodám dané komponenty a volat je. Metoda saveUser je vyvolaná pomocí action, tudíž musí vracet strukturu, která pak může být použita pro cíl navigace. V tomto případě tato metoda vrací String, který je mapován na url, na kterou se po provedení kódu této metody přesměruje webové rozhraní. Metoda cancel se ale využívá v actionListeneru, nemá návratový typ a nikam nepřesměrovává.

```
<p:commandButton value="#{msg.save}" action="#{custBean.saveUser}" />
<p:commandButton value="#{msg.cancel}" immediate="true" actionListener="#{custBean.cancel}" />
```

Výpis 3: Ukázka JSF buttonu

4.1.3 PrimeFaces

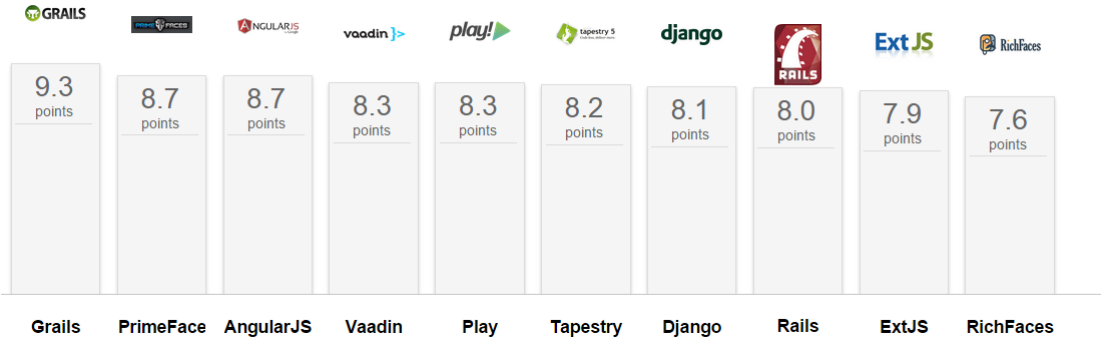
PrimeFaces je knihovna, tvořená jedním jar bez dalších dependencies. Dovolím si citovat myšlenku, která stála za vznikem PrimeFaces:

Components in PrimeFaces are developed with a design principle which states that "A good UI component should hide complexity but keep the flexibility"while doing so. [4]

Což by se dalo volně přeložit asi takto: Komponenty v PrimeFaces jsou vyvíjeny s návrhovým principem, který říká, že "Dobrá UI komponenta by měla nejen skrývat složitost, ale také udržovat pružnost."s tím, že to také dělají.

PrimeFaces jsou hodnoceny jako jedny z neoblíbenějších frameworků z hlediska developerů, které poskytují vytváření bohatých uživatelských rozhraní v jazyce Java.

DevRates.com zveřejnil tuto statistiku:



Obrázek 10: Oblíbenost frameworků developery, převzato ze zdroje [11]

Více informací a konkrétní hodnocení můžete nalézt zde [12].

PrimeFaces mají opravdu bohatou škálu komponent umožňujících různá využití, od často používaných selectMenu a button až po grafy a sockety. Všechny komponenty včetně krátkých tutoriálů popisuje zdroj [4].

4.1.4 Representational state transfer

Representational state transfer (dále REST) je způsob, jak pro komunikaci využívat jednoduchá HTTP volání. Je reprezentován datově a používá se pro snadný přístup k datům (resources). REST neslouží pouze pro webové služby, ale nachází uplatnění tam, kde je potřeba jednoduchým způsobem zaslat požadavek a dostat zpět data určitého formátu, jako například text, obrázek atd.

Některé aplikace sice využívají REST, ale nejsou tzv. RESTfull, to znamená, že nemají jeho plnou podporu.

REST funguje na základě požadavku (tzv. resource), který má svůj vlastní identifikátor, což může být například URL. Reprezentace požadavku už je individuální, nejčastěji jde o XML, JSON nebo HTML. Dalšími formáty jsou například PDF nebo SVG. Klient nepracuje přímo se samotným požadavkem, ale má k dispozici jeho reprezentaci.

Sémantika RESTu je omezená, tvoří ji pouze CRUD operace. Není zde například možné používat sémantiku, která by odpovídala nějakým volaným metodám. Také se oproti jiným metodám, jako například RPC (Remote Procedure Call) získává stav aplikace určený daty, ale ne chování aplikace. Typickým rysem REST je i jeho bezestavovost,

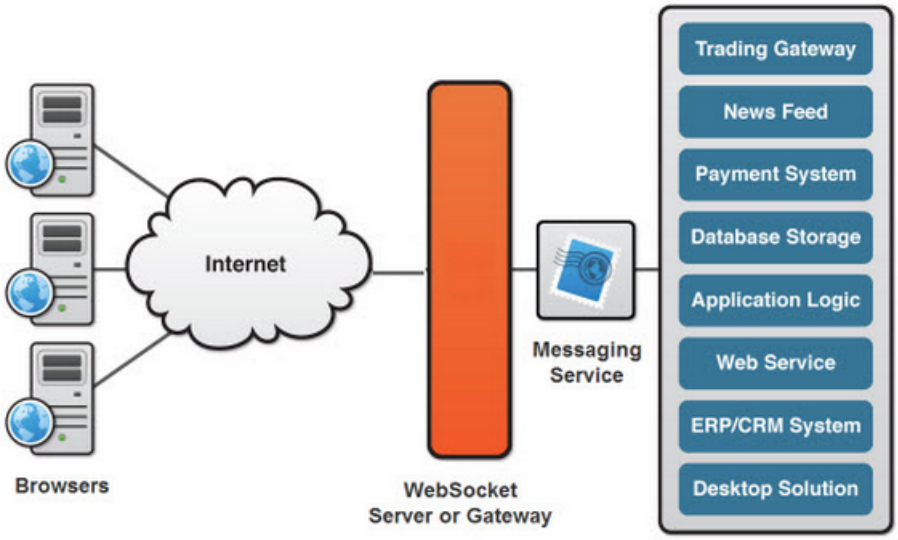
což znamená, že jeden požadavek musí obsahovat všechny informace nutné k jeho vykonání.

4.1.5 WebSocket

Specifikace HTML5 WebSocketů určuje API, které umožňuje webovému rozhraní oboustrannou komunikaci s remote hostem. Je něčím, co by se dalo nazvat jako další evoluční krůček v porovnání například s Ajaxem.

Poskytuje rozhraní a definuje oboustrannou komunikaci pro každý socket. Výhodou je, že není nutné udržovat dlouhé spojení. To s sebou nese řadu výhod, například se takto snižuje množství nepotřebné síťové komunikace, narozdíl od dlouho trvajících spojení, která měla simulovat full-duplex tak, že udržovala dvě spojení.

Základní architekturu technologie WebSocket můžeme vidět na obrázku 11



Obrázek 11: Architektura WebSocket

Tato technologie může být tedy někdy výhodnější než technologie zvaná heartbeat, která se v určitých časových intervalech ptá serveru, zda má nová data. Odpověď je totiž většinou negativní, a tak dochází ke zbytečné komunikaci a zatížení sítě a prostředků. V takových případech je WebSocket jednoznačně vhodnější technologií.

Z hlediska implementace WebSocketů exituje spousta možností. Programátor si může napsat vlastní WebSocket server a poslouchajícího klienta. Technologie WebSocket je ale oblíbená a často používaná, takže existují frameworky a knihovny, které práci s touto technologií značně ulehčují.

4.1.6 Enterprise JavaBeans

Specifikace EJB popisuje třídy a rozhraní, které definují vztah mezi enterprise beans a klientem a mezi enterprise beans a EJB konteinerem. EJB obstarává logickou vrstvu projektu.

EJB využívá aplikační kontejner. Z toho důvodu je nutné volit server, který tedy musí podporovat EJB technologii. Aplikační kontejner umí nejen vytvářet instance tříd EJB komponent, ale také je uchovávat a vracet, což šetří místo - nemusí se znovu vytvářet. Je tedy mnohem výhodnější využívat EJB injection než například dědičnost.

Java Beans

Starší verze @EJB nutila programátora, aby každá Java Bean implementovala rozhraní buď Local nebo Remote - podle toho, zda jsou využívány jen lokálně, nebo je využívá i jiná JVM. Nyní stačí pouze anotace @Local (chování jako interface Local). Také anotace @ManagedBean je již deprecated. Tuto anotaci nahradila anotace @Named s parametrem, který udává název komponenty, díky kterému je pak dostupná přímo ze stránek.

V ukázce zdrojového kódu 5 vidíme použitou deklaraci @Named a @RequestScoped. Dále jsem použila anotaci @EJB, která mi pomohla provést EJB injection na třídy DatabaseService a LogService, které jsou @Stateless. Dále je zde použita @ManagedProperty, která umožní použít stejnou instanci třídy, která je uložena v jiné komponentě.

```
@Named("administration")
@RequestScoped
public class AdministrationBean implements Serializable {

    @EJB
    DatabaseService databaseService; //= lookupDatabaseServiceBean();

    @EJB
    LogService logService;

    @ManagedProperty(value = "#{sessionTracker.user}")
    private AppUser user;
...
}
```

Výpis 4: Ukázka použití Java Bean

Novinky v EJB pro Java EE 7

Java EE 6 s sebou přináší spoustu novinek, jako třeba @Singleton, @Asynchronous, @Schedule, Portable JNDI name, EJBContainer.createEJBContainer, EJB 3.1 Lite a dalších. Java EE 7 už tolik novinek nepřináší, na druhou stranu ale novinky z Java EE 6 upřesňuje a slouží spíš jako bližší specifikace.

Více o novinkách se můžete dozvědět v dokumentu, který je ke stažení ve zdroji [9].

4.1.7 Contexts and Dependecy Injection

Contexts and Dependecy Injection (CDI) je technologie, jejíž hlavním cílem je jednoduché propojování s dalšími částmi aplikace.

Základní funkcí CDI je **Context**, s kterým je možno definovat životní cyklus, a **Dependency Injection**, které umožňuje injektování komponent v aplikaci způsobem, který je typově bezpečný a až v době, kdy probíhá deploy, se vybírá, která implementace se bude injektovat.

CDI umožňuje některým komponentám a JSF ManagedBeanám, aby byly injektovány a mohly volně komunikovat pomocí vyvolávání a obsluhy událostí. Také umožňuje Java EE komponentám, jako jsou Servlet nebo Bean, aby jejich životní cyklus byl úzce spjat s životním cyklem aplikace. K tomu účelu poskytuje různé nastavení logické délky životnosti, tzv. scope.

4.1.8 Porovnání CDI a EJB

CDI a EJB mohou vyvolávat dojem, že obsahují totožnou funkcionalitu. V této kapitole je vysvětleno, v čem je rozdíl a proč se někdy používá CDI a jindy zase EJB.

CDI jako takové umožňuje inject, scoping a event bus. EJB oproti tomu umožňuje dependency injection, declarative transactions, declarative security, pooling, concurrency control, asynchronous execution a remoting.

EJB nemůže využívat event bus tak, jako CDI, ale má speciální typ Java Bean, který umožňuje poslouchat zprávy - message driven bean.

Z toho vyplývá, že EJB jako takové mají spoustu funkcí, které nemohou být lehce nahrazeny CDI. Na druhou stranu, CDI a EJB mohou být používány dohromady.

Obě technologie ale umožňují injection, každá ale s trochu jinou myšlenkou. CDI umožňuje injektovat implementační rozhraní kamkoli. Objekt, který je injektován, nemusí být jen EJB. CDI může být použito, když je zapotřebí injektovat servisy, které nejsou EJB, jinou implementaci nebo algoritmy. V tomto ohledu je tedy mnohem flexibilnější, než EJB.

EJB, ve spojení s @EJB anotací, také umožňuje inject. Hlavní myšlenkou ale je, že třída, ve které se @EJB inject používá, je spravována EJB kontejnerem.

Inject pomocí obou metod pro porovnání:

```
@EJB EJBService ejbService;
@Inject EJBService ejbService;
```

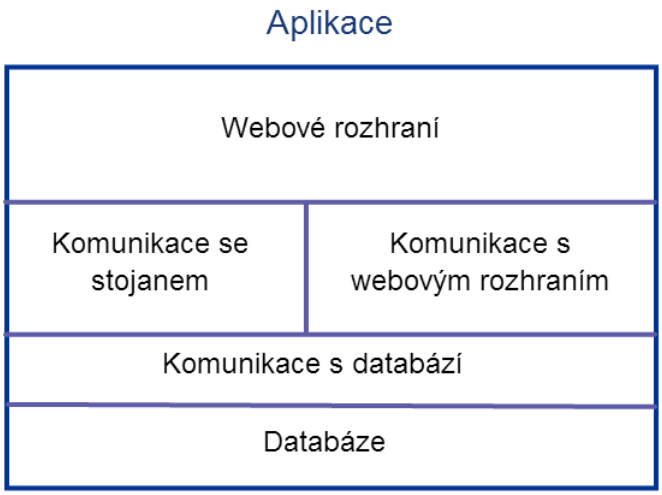
Výpis 5: Ukázka použití Java Bean

4.2 Struktura aplikace

Vzhledem k tomu, že stojan a jeho komunikace nebyla změněna, další části se budou zabývat převážně aplikaci. Ta byla původně tvořena jako jediný projekt Web Application, zahrnující všechny části - od komunikace s databází až po webovou prezenční vrstvu. Cílem tedy bylo ji rozvrstvit na části tak, aby se každá část starala o určitou funkcionalitu. Taková aplikace bude nejen mnohem lépe přizpůsobovat dalším změnám, které mohou být u tohoto systému provedeny v budoucnosti, ale také bude mnohem přehlednější z programátorského hlediska.

Jedním ze způsobů, jak docílit žádaného rozvrstvení, je vytvoření Enterprise projektu, který umožňuje rozdělení na ejb modul a webový modul. Ejb modul se stará o databázový přístup a zpracovává data, která se mají do databáze uložit. Webový modul je pak ta část, která obsahuje jak samotné view, tak controller, který zpracovává data načtená a poslaná ejb modulem a pak je posílá k vykreslení do jednotlivých view tříd.

Další možností je vytvoření rovnou celého Maven projektu, který má v zásadě podobnou strukturu, jako Enterprise projekt, ale navíc usnadňuje práci v jednotlivých projektech s ohledem na externí knihovny a má další výhody oproti Enterprise projektu. Z toho důvodu byla aplikace psaná jako Maven projekt. Struktura aplikace vypadá tak, jak ukazuje obrázek 12. Webová část tedy využívá JavaBean pro svou logiku a kombinaci css, html, JSF a JSP pro vykreslování webových stránek.



Obrázek 12: Základní schéma struktury aplikace

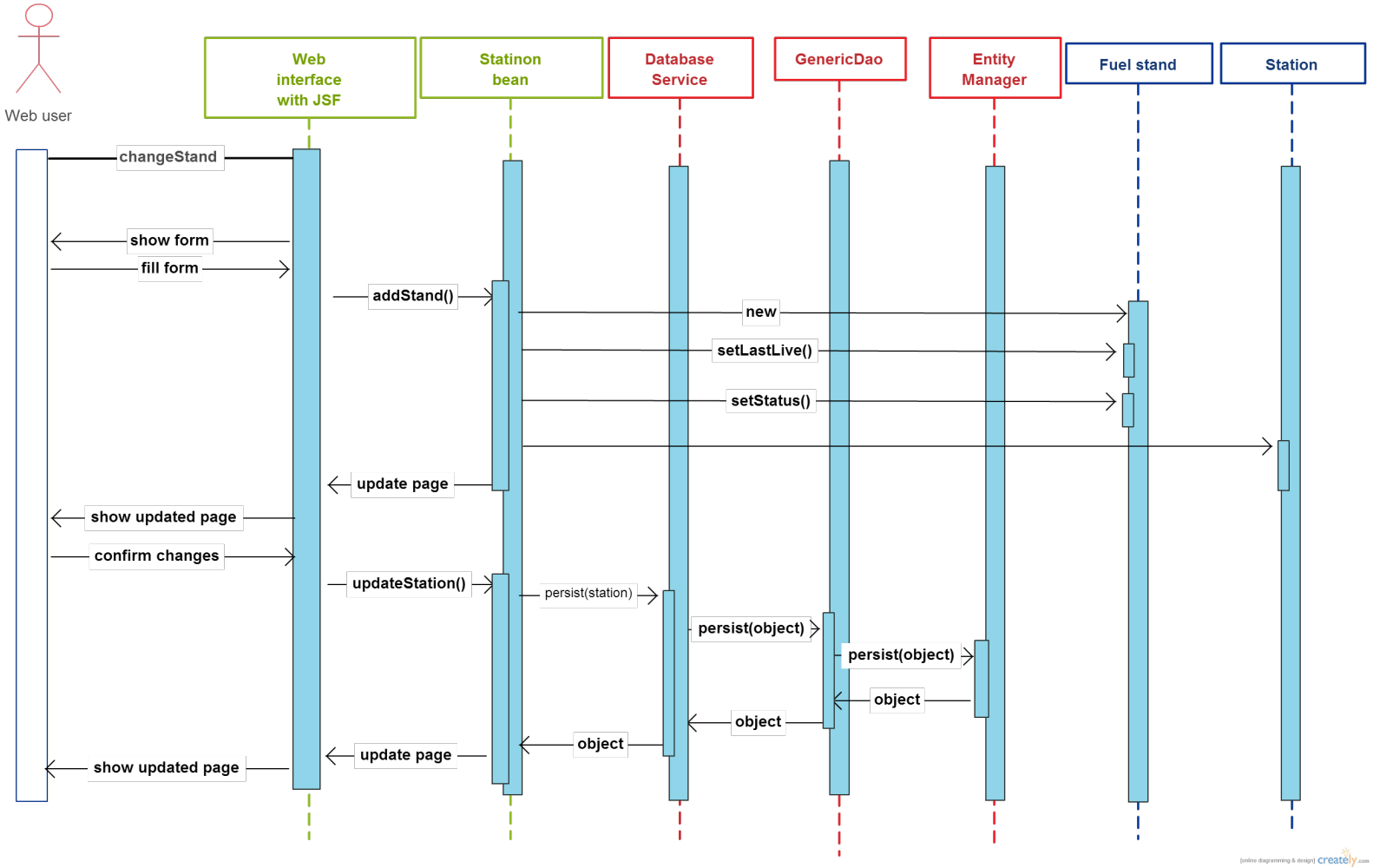
4.2.1 Struktura databáze

Již existuje databáze, která byla používána původním systému. Vzhledem k zjednodušení přechodu na nový systém by bylo vhodné strukturu samotné databáze ponechat tak, jak již byla naimplementována. Schéma této databáze je viditelné na obrázku 6.

4.2.2 Struktura části pro komunikaci s databází

Původně vrstvy aplikační logiky komunikovali přímo s vrstvou, která obstarávala data z databáze. Chyběla zde jakási servisní vrstva, která by zprostředkovávala komunikaci a zachovala zapouzdření. Aby byl tento nedostatek odstraněn, byl použit vzor DAO [7], který tento problém elegantně řeší. Použití návrhového vzoru DAO v programu je patrné v sekvenčním diagramu 13.

Příklad komunikace s uživatelem ukazuje sekvenční diagram 13. Zobrazuje vkládání nového stojanu a jeho přiřazení ke stanici, ke které patří.



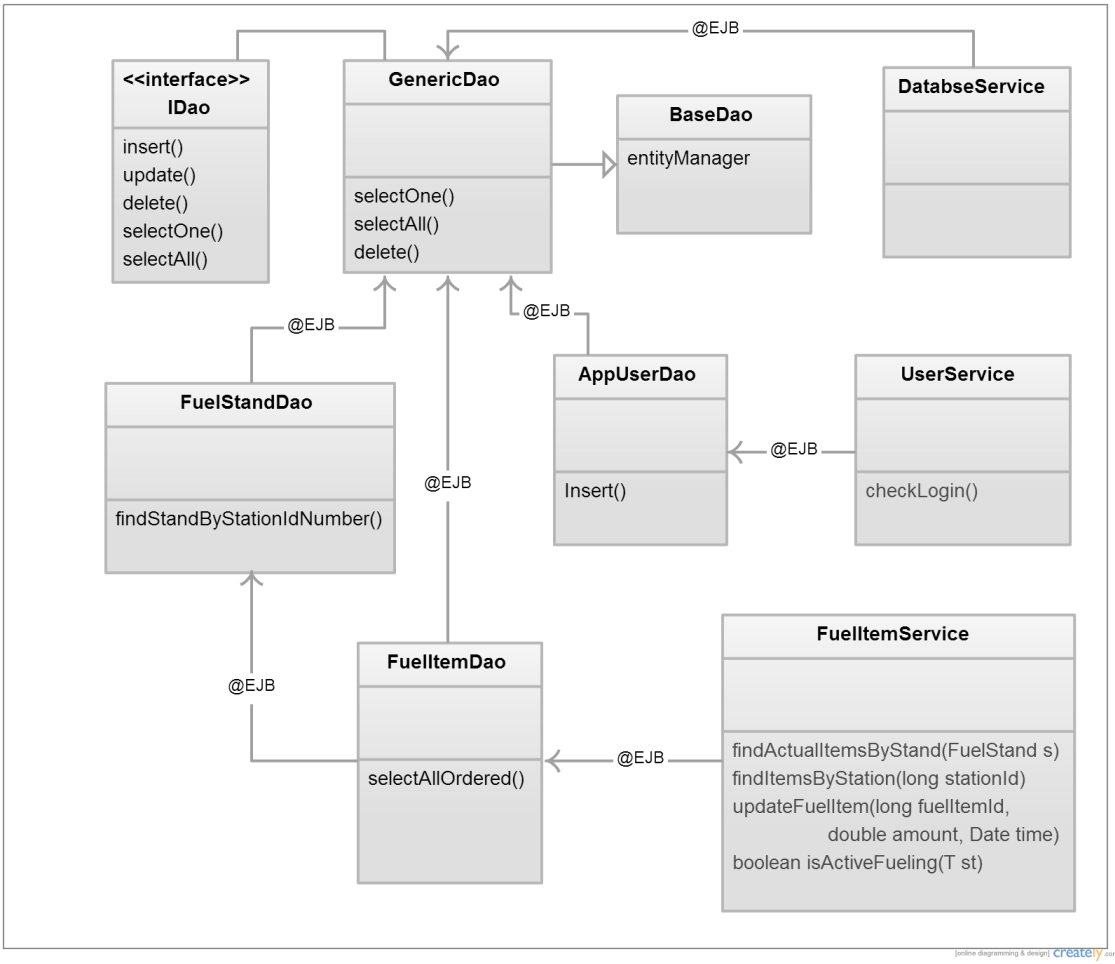
Obrázek 13: Sekvenční diagram pro vložení stojanu

Aplikace využívá komponenty Entity Beans k reprezentaci dat v databázi. Pro přístup do databáze využívá každá entita funkcí CRUD.

Z hlediska implementace byl přístup do databáze rozložen na část, která je společná všem entitám a obsahuje základní operace nad databází. Dále určité entity potřebují používat vlastní přístup či specifické operace nad daty. K tomu slouží další třídy, vlastní těm entitám, které je potřebují. V těchto třídách se nachází pouze implementace specifických operací.

Hlavní třída, společná všem entitám, implementuje generické funkce a je chápána jako hlavní prvek při přímé komunikaci s databází. Na obrázku 14 můžeme vidět, že tu funkci zastává třída GenericDao.

Servisní třídy kopírují Dao třídy, s kterými komunikují pomocí @EJB injection. Na obrázku 14 vidíme stručné znázornění struktury databázové části.



Obrázek 14: Databázová část programu

4.2.3 Struktura komunikace stojanu a serveru

Mezi stojanem a serverem probíhá komunikace pomocí RESTu, který je blíže popsán v kapitole 4.1.4. Stojan volá server pomocí url, která je vytvořená pomocí schématu:

http://server:port/Aplikace/ROOTIdentifikatorRESTu/RESTPozadavek?parametry

Ukázku javovské třídy, která se stará o komunikaci skrze REST, můžeme vidět na ukázce 6.

```
@Path("/ns-itu/authenticate")
@Stateless
public class AuthenticateCommunication {
    ...

    @GET
    @Produces("text/plain")
    public Response authenticate(
            @QueryParam("hash") String hash,
            @QueryParam("pin") String pin,
            @QueryParam("stationId") int stationId,
            @QueryParam("number") int number
            ) {
        ResponseBuilder responseBuilder;
          ...
        return responseBuilder.build();
    }
}
```

Výpis 6: Ukázka třídy pro REST komunikaci

@Path definuje url, které tato třída poslouchá. Díky tomu se může každá třída soustředit pouze na tu část komunikace, která je pro ni zajímavá. Tímto způsobem jsou odděleny url pro přihlášení uživatele nebo pro nabíjení ze stojanů.

@QueryParam jsou příchozí parametry z REST požadavku, které si uložíme do vlastních proměnných, se kterými se dále pracuje.

ResponseBuilder je třída, která umožňuje tvořit odpověď, tzv. response, která se pak odesílá jako odpověď na požadavek, tzv. request, zde pomocí metody GET.

Příkazem .build() se vytváří instance Response, která se generuje z aktuálního ResponseBuilderu. Builder se pak vrací do prázdného stavu.

Vlastnosti odesílané odpovědi můžeme nastavovat buď přímo metodami, které obsahuje ResponseBuilder, nebo přes třídu Response, a to takovýmto způsobem:

responseBilder = Response.status(Response.Status.FORBIDDEN);

V případě, že v průběhu metody nastane chyba, je odchycena pomocí try/catch, způsobem, jaký uvádí kód 7.

```
catch (ParseException ex) {
            return Response.serverError().build();
        }
```

Výpis 7: Vyvolání výjimky ve zpracování REST požadavku

Dále zde naleznete konkrétní informace o posílaných zprávách a třídách, které je zpracovávají. Také jsou zde uvedeny ukázky kódů. Vstupními parametry se myslí příchozí parametry přes definovanou metodu. Ukázku parametrů můžete vidět v kódu 6.

Hlavní třída, která definuje všechny další třídy související s přímou komunikací, je **VoltageApplication**. Její kód je uveden v ukázce 8. Definuje hlavní uzel komunikace a dědí přímo z Aplikace.

```
@ApplicationPath("/ns-itu/*")
public class VoltageApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        final  Set<Class<?>> classes = new HashSet<>();
        classes.add(AuthenticateCommunication.class);
        classes.add(FuelingCommunication.class);
        classes.add(HeartBeatCommunication.class);
        classes.add(LoggingCommunication.class);
        return classes;
    }
}
```

Výpis 8: VoltageApplication

Nyní následuje popis tříd, které jsou uvedeny ve VoltageAppliaction.

Hned první je třída **AuthenticateCommunication**. Konkrétní informace jsou uvedeny v tabulce 3. Tato třída vrací v objektu Response informace o průběhu komunikace, který vidíte v ukázce 9. AuthState je informace o průběhu autentifikace. Může nabývat hodnot OK, Fail nebo Denied.

```
StringBuilder  myResponse = new StringBuilder();
myResponse
   .append(AuthState.OK)
   .append(DELIM)
   .append(fuelUser.getId())
   .append(DELIM)
   .append(price)
   .append(DELIM)
   .append(fuelUser.getJmeno().getCeleJmeno());
ResponseBuilder responseBuilder = Response.ok(myResponse.toString(), MediaType.
     TEXT_PLAIN_TYPE);
return responseBuilder.build();
```

Výpis 9: Struktura Response pro AuthenticateCommunication

FuelingCommunication obstarává komunikaci týkající se samotného nabíjení ze stojanu. Konkrétní informace naleznete v tabulce 4. Vstupem i výstupem je plain text. Zde probíhá ověření tzv. flagu, který nese informace o stavu nabíjení. Nabývá hodnot START, CONTINUE, END a ERROR. V případě, že je flag START, vytvoří se do databáze nový

AuthenticateCommunicatoin		
používaná metoda:		GET
obsluhovaná url:		/ns-itu/authenticate
vstupní parametry:		
hash	String	hash uživatele
pin	String	pin uživatele
stationId	int	id stanice, ze které se uživatel přihlašuje
number	int	výstup stojanu

Tabulka 3: Parametry třídy AuthenticateCommunitacion

záznam o nabíjení. V případě CONTINUE i END se aktualizuje záznam v databázi. Response vrací informace o průběhu - pokud nenastal problém, vrací se status OK, jinak serverError().

FuelingCommunication		
používaná metoda:		PUT
obsluhovaná url:		/ns-itu/fueling
vstupní parametry:		
fuelItemId	long	id záznamu
userID	long	id uživatele
stationId	int	id stanice
number	int	číslo výstupu stojanu
outputId	int	id výstupu
time	String	čas dobíjení
amount	double	množství energie
flag	FuelState	stav stojanu

Tabulka 4: Parametry třídy FuelingCommunication

HeartBeatCommunication je další třída, komunikující se stojanem. Ta slouží k tomu, aby o sobě mohl stojan pravidelně dávat vědět, zda je takzvaně živý, nebo ne. V případě, že se neozve po určitou dobu, je jeho stav změněn z CONNECTED na DISCONNECTED. Další informace uvádí tabulka 5. Do databáze se ukládá pokaždé čas, kdy o sobě dal stojan vědět.

LoggingCommunication je třída obstarávající zaznamenávání provozních informací. Příkladem může být přihlašování stanice do aplikace. Když se stanice přihlásí, je vytvořen nový FuelLogItem, do kterého se uloží data s informacemi o stanici, stojanu, času přihlášení a text a uloží se do databáze. Konkrétní informace jsou v tabulce 6.

HeartBeatCommunication		
používaná metoda:		GET
obsluhovaná url:		/ns-itu/heartbeat
vstupní parametry:		
stationId	int	id stanice
number	int	číslo výstupu stojanu
status	FuelStandStatus	status stojanu

Tabulka 5: Parametry třídy HeartBeatCommunication

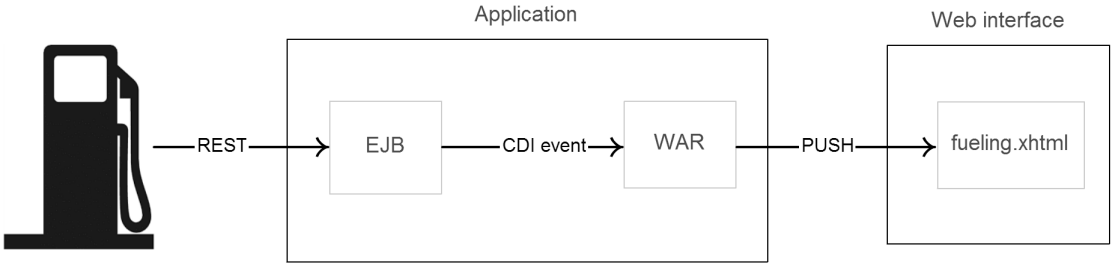
LoggingCommunication		
používaná metoda:		PUT
obsluhovaná url:		/ns-itu/logging
vstupní parametry:		
stationId	int	id stanice
number	int	číslo výstupu stojanu
time	String	čas zalogování
text	String	popis zalogované události

Tabulka 6: Parametry třídy LoggingCommunication

4.2.4 Struktura komunikace serveru s klientskou aplikací

V případě, že na stanici přijede zákazník a začne si dobíjet svůj elektromobil, začne vysílat data do aplikace. Toto by měl být impulz pro aplikaci, že by měla aktualizovat data u daného stojanu a aktualizovat webovou stránku, aby uživatel mohl vidět, že právě probíhá dobíjení.

Způsob této komunikace včetně nástinu implementace ukazuje schéma 15.



Obrázek 15: Způsob komunikace server - webové rozhraní

Stojan začne posílat data do aplikace. Tato data jsou odchytávány v EJB části aplikace, která je zpracuje a vyvolá událost, která je odchytávána ve webovém modulu projektu. Při odchycení této události se pomocí PrimeFaces tagu p:socket vyvolá javascriptový kód, který aktualizuje data o tom stojanu, z kterého začalo probíhat dobíjení.

Třída FuelingCommunication (blíže popsána v předchozí sekci) je informována v případě, že začne dobíjení. Pak vyvolá událost, jak vidíme v ukázce 10.

```
switch (flag) {
                case START: {
                    Date date = sdf.parse(time);
                    fuelItem = fuelItemDao.createFuelItem(userId, stationId, standId,
                     outputId, amount, date);
                    statusEvent
                     . fire (new ChargingEvent(standId, stationId, outputId, "CHARGING"));
                    break;
                }
                 ...
}
```

Výpis 10: Vyvolání události ve FuelingCommunication

Do třídy StatusEvent jsou uloženy informace o stojanu - v tomto případě jeho Id.

Tato událost je odposlouchávána třídou FuelingBean. Metodu, která se o odposlouchávání stará, můžete vidět v ukázce 11.

```
public static void onStatusEvent(@Observes StatusEvent statusEvent) {
        EventBus eventBus = EventBusFactory.getDefault().eventBus();
        eventBus.publish("/status", statusEvent.getFuelStand());
    }
```

Výpis 11: Odposlouchávání události třídou FuelingBean

Pomocí anotace @Observers dochází k odposlouchávání události. Data ze třídy StatusEvent jsou uloženy do proměnné statusEvent a dále jsou zpracovávány. Pomocí EventBus se posílá metodou .publish() na kanálu "/status"instance třídy FuelStand, která je dále odchycena třídou oanotovanou jako @PushEndpoint, viz ukázka 12.

```
@PushEndpoint("/browser")
public class BrowserStatsResource {

    @OnMessage(encoders = {JSONEncoder.class})
    public String[] onMessage(FuelStand data) {

        String[] string = new String[2];
        String id = "id" + Long.toString(data.getStation().getId()) + data.getId();
        string[0] = id;
        string[1] = data.getStatus().toString();

        return string;
    }
```

Výpis 12: Třída poslouchající na kanálu browser

Odsud jsou už data posílána pomocí socketu. Ten je odchycen na stránce fueling.xhtml pomocí PrimeFaces atributu p:socket, jak ukazuje kód 13. Ten určuje, která javascriptová metoda příchozí data zpracovávat.

```
<p:socket onMessage="changeStatus" channel="/status" />
```

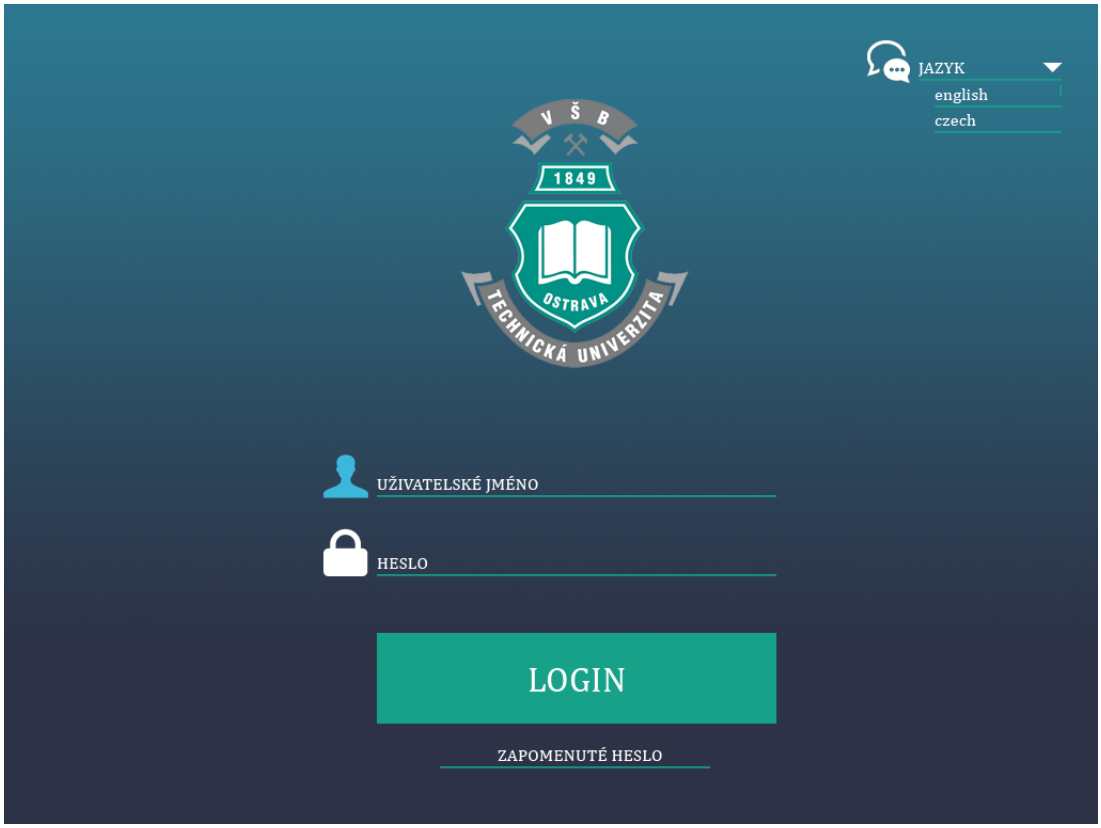
Výpis 13: Odchycení socketu stránkou fueling.xhtml

V případě, že přijde zpráva na kanálu "/status", je vyvolána javascriptová metoda changeStatus, která se stará o aktualizaci ikony u daného stojanu.

Pro aktualizaci a zobrazování záznamů o tom, že je ze stojanu dobíjeno, slouží třída UpdateFuelBean.

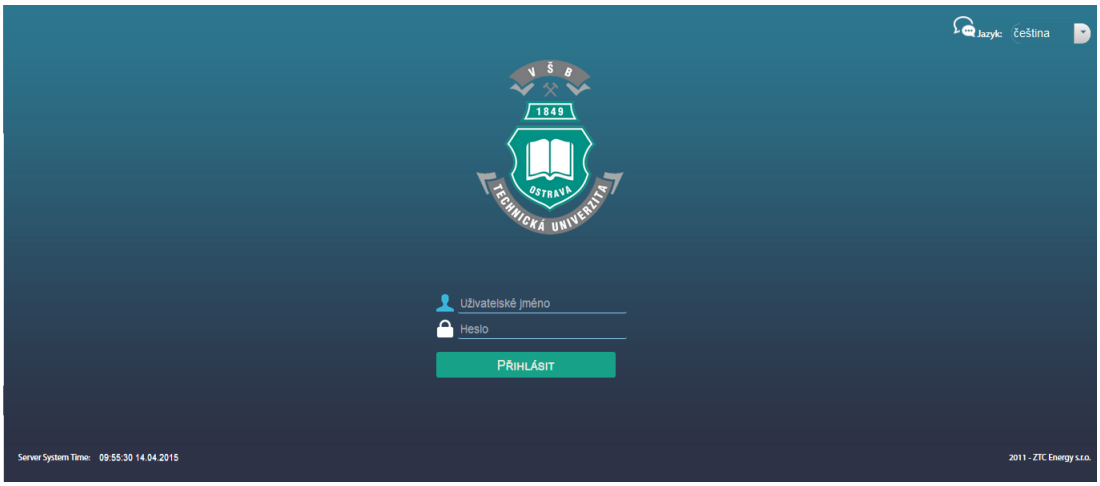
4.3 Změny ve webovém prostředí

Na obrázku 16 můžeme vidět, jak by měl vypadat nový vzhled webových stránek. Tento návrh dodala osoba pověřená konzultantem práce.



Obrázek 16: Návrh designu

Současný vzhled můžeme vidět na obrázku 17. Do budoucna se počítá s dalšími úpravami, aby design co nejvíce odpovídal představám zákazníka.



Obrázek 17: Současná grafická podoba webových stránek

5 Závěr

Cílem této práce bylo vylepšit současný systém pro monitorování dobíjecích stanic elektromobilů za pomocí nových technologií. Jako programovací jazyk této aplikace byla použita Java. Funkci aplikačního serveru zastal GalassFish.

Požadavky zadání, jako použití nových technologií a rozdělení aplikace na logické části, byly splněny. Grafické rozhraní není ještě zcela dokončeno. Na aplikaci se dále bude pracovat a pokud bude dodán kompletní grafická návrh, bude dokončena v průběhu května či června. V budoucnu se aplikace bude rozrůstat a přizpůsobovat novému rozšíření mezi další uživatele.

6 Reference

[1] Oracle: Hardware and Software, Engineered to Work Together.
[online]. [cit. 2015-05-01].
Dostupné z: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[2] Oracle: Hardware and Software, Engineered to Work Together.
[online]. [cit. 2015-05-01].
Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview-140548.html

[3] WebSocket.org: A WebSocket Community. [online]. [cit. 2015-05-01].
Dostupné z: https://www.websocket.org

[4] PrimeTek Informatics. PrimeFaces. [online]. 2009-2014 [cit. 2015-05-01].
Dostupné z: http://www.primefaces.org

[5] Apache OpenJPA Userś Guide: Entity Lifecycle Management.
[online]. [cit. 2015-05-01].
Dostupné z: http://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa_overview_em_lifecycle.html

[6] Oracle: Lesson: Generics. [online]. [cit. 2015-05-01].
Dostupné z: https://docs.oracle.com/javase/tutorial/java/generics/

[7] Oracle: Core J2EE Patterns - Data Access Object. [online]. [cit. 2015-05-01].
Dostupné z: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

[8] Oracle: Entities - The Java EE 6 Tutorial. [online]. [cit. 2015-05-01].
Dostupné z: http://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html

[9] The Java Community Process(SM) Program: JSR-000345 Enterprise JavaBeansTM 3.2. [online]. [cit. 2015-05-01].
Dostupné z: https://jcp.org/aboutJava/communityprocess/final/jsr345/index.html

[10] Help - Eclipse Platform. [online]. [cit. 2015-05-01].
Dostupné z: http://help.eclipse.org/indigo/index.jsp

[11] DevRates: Open source reviews by real users. [online]. [cit. 2015-05-01].
Dostupné z: http://devrates.com/stats/top?tagName=web+framework

[12] DevRates. [online]. [cit. 2015-05-01]. Dostupné z: http://devrates.com/project/list